Truck Platooning: week 9

Changyao Zhou, Jiaxin Pan

February 22, 2022

1 Introduction of the project

The goal of this project is to chase a chosen target vehicle by predicting the steering angle and throttle input of the ego-vehicle, given the velocity of both vehicles and images taken from cameras.

The control values (steering angle and throttle) computed by Model Predictive Control (MPC) are used as ground-truth. MPC is an advanced method of process control that is used to control a process while satisfying a set of constraints[WIK]. It takes a prediction horizon instead of a single time step into account, and aims to get an optimal control result minimizing the objective function within the prediction horizon. For learning the MPC controller, we take [MPC] as reference.

The project is designed to be realized in the following steps:

1). Control the ego-vehicle with Model Predictive Control (MPC) in the Carla simulator

2). Given x, y-location, yaw angle and velocity of both vehicles, train a fully connected neural network to control ego-vehicle by using the MPC output as ground-truth.

2). Given object velocity, ego-velocity and RGB image, train a neural network to control the ego-vehicle by using the MPC output as ground-truth

3). Further improvement by changing the input of the neural network

2 Overview of this week

Last week we trained the CNN-MLP model and the end-to-end CNN model and got desirable results from both. This week we further continued with the CNN-MLP model, but this time with images sampled from image rendering instead of directly from Carla simulator.

The largest problem of image rendering, which is the black hole on the edges, is almost solved and we got a fairly good result both for the image rendering and the model training.

All results of models mentioned above were tested with the 'online' mode in CARLA, where the target vehicle is set in 'autopilot' mode. All the videos are stored under https://syncandshare.lrz.de/getlink/fi58hQcAmYS6UdhxDbAFx8VH/week9.

3 Training with rendered images

3.1 Dataset Preparation

Last week we tried to generate images with image rendering based on the on-trajectory images sampled from the Carla simulator, however, the holes were extremely large, which makes the rendered images look very different from the real images observed by the camera in Carla simulator. In this week, we tried to fix this problem.

Instead of collecting square images (750x750) with the camera fov (Field of view) 100°, we collected rectangular images (800x300) with a fov of 130° first for on-trajectory routes. With the rectangular source image, even with large lateral offset (2m), the holes on edges only takes up a small part of the whole image, as can be seen in Figure 1. To fix these black parts, we first crop the rectangle image from the top and get a smaller image with size 250x400. Then the cropped image is resized to 128x128, which is the same size as the CNN input.

Figure 1: Lateral Offset 2m

Figure 2: Longitudinal Offset -3m

The holes with lateral offset is in this way fixed, however, the problem with longitudinal offset is even more severe. Many pixels in the bottom of the image are missing with a negative longitudinal offset. As can be seen in Figure 2, the holes on the bottom cover almost one third of the length of the image. On the other hand, there is little holes in the top half of the image, since the depth of the pixels representing the sky is very large and the offset is much smaller in comparison to the depth, which leads to little change on the projected positions of these pixels.

Due to the fact that most of the holes are in the bottom, instead of center cropping the images, we cropped the bottom part of the image and the upper part is reserved, in order to reduce the portion of the missing pixels. The image size is reduced to 400x250, with which most of the missing parts is cropped.

The images are fed to the CNN model as inputs, together with the velocities of the two vehicles, the outputs are Δx, Δy, Δyaw between the target- and ego vehicles. Since the input size of the CNN model we trained was 128x128, the images are resized to 128x128. The final images are shown in Figure 3-6.

The delta information is computed with:

$$T_B^C = T_A^C T_B^A \tag{1}$$

where T is the transformation matrix, A, B and C represents the original ego vehicle, target vehicle and deflected ego vehicle respectively. In this way we can compute the relative transformation of the target vehicle w.r.t. deflected ego vehicle T_B^C.

Figure 3: Rotational offset 10°

Figure 4: Lateral Offset 2m

Figure 5: Longitudinal Offset +2m

Figure 6: Longitudinal Offset -3m

3.2 Training the CNN-MLP Model

We used the same architecture of CNN as in last week, as can be seen in Table 1.

We first collected 8,000 on-trajectory images from 8 routes in Town 03 and Town 04, and generated 40,000 images in total, with only lateral offsets and rotations for 32,000 off-trajectory images. After training, it turns out that the model could predict right values by turning but not accelerating, which means that the longitudinal offsets are still needed for generating off-trajectory images. What's more, we deleted several trajectories with constant speeds and added more trajectories with sudden accelerations.

The new dataset has 45,500 images from 8 trajectories in total, where 39,000 are off-trajectory images with lateral, longitudinal and rotational offsets. With training of 75 epochs, the validation loss for Δx, Δy, Δyaw was 1.2e-2, 6.6e-3 and 5e-4 respectively.

The losses were extremely small, even smaller than the losses from the CNN model of last week, which was trained with a larger dataset of 60,000+ images, especially the loss of Δyaw, which was 1e-3, 2 times larger than the loss of this week. The reason might be that the model learns the Δyaw value from the missing parts behind the target vehicle, which are caused when generating off-trajectory images with lateral offsets. The missing parts form a shape similar to the vehicle and always has the same direction as the lateral offset. When having the ego vehicle on the right side of its original position, the missing parts are also on the right side of the target vehicle, vice versa.

This is not the way what we supposed the network to learn the Δyaw value, since the missing parts can not be observed in the real images. Despite the fact that the model may learn the Δyaw value in an unexpected way, the model does perform well when we did inferences in Carla simulator on different trajectories in different towns. All the videos are stored under https://syncandshare.

Table 1: CNN Architecture

Layer Number	Layer Type	Layer Input	Layer Output
1 - 13	feature extractor from pretrained Alexnet	3x128x128	256x3x3
14	MaxPooling	256x3x3	256x1x1
15	Flatten	256x1x1	258
16	Linear Layer	258	64
17	ReLU	64	64
18	Linear Layer	64	3

lrz.de/getlink/fi58hQcAmYS6UdhxDbAFx8VH/week9.

As can be seen in the videos, the model can predict desirable values for different unseen trajectories in different towns. Even if there is acceleration, the model can handle it (CNN_MLP_town04_unseen1.mp4).

By sharp turns, the model is not that stable as the model in the last week, the ego-vehicle will be shaky when turning (13" in CNN_MLP_town03_unseen2.mp4). But despite the shaky problem, the ego vehicle can follow the target vehicle in a tolerant way. In addition, the ego-vehicle might also take a closer shortcut route and reach the sidewalk by a sharp right angle turn. Then the ego-vehicle will eventually return to the normal route. (3s in CNN_MLP_town05_unseen2.mp4) The probable reason might be that the images observed by reference does not have the black part behind the target vehicle, which the network learned the Δyaw value from, thus the error of predicting Δyaw value is larger than expected. In town 05, which is not covered in the training dataset, the model can predict right control values, even if the background is brand new to it.

In conclusion, the CNN-MLP model trained with the rendered images performs well by the vehicle following problem, but it may lead to a shaky ego vehicle especially by turning, which makes it not as perfect as the CNN-MLP model trained with images collected in Carla simulator. The result above can prove that this method for data collection based on image rendering can also be realized in practice. We only need to collect the on trajectory data in the real world, including the driving states data and the rectangle images with larger width and a larger FOV. Then ideally we could automatically generate images for all possible offsets, which can be seen as data augmentation. Doing so eliminates the tedious collection process for off-trajectory data and its risks in practice.

4 Conclusion and future work

This week we managed to generate off-trajectory images with lateral, longitudinal and rotational offsets from on-trajectory images with image rendering method. The CNN-MLP model trained with this dataset has a fairly good performance. We did inferences in different towns with different routes and the ego vehicle could follow the target vehicle in most of the cases. The model is however not that stable in comparison to the model trained with the images collected directly from the Carla simulator, since the ego vehicle is more shaky especially by turning.

Till now, we have tried different models and different datasets to train our model. But we haven't evaluated the performances of these models formally, so, in the next step, we will try to evaluate the performance. If there is time left, maybe we will try out with the RNN to predict the velocities of the both vehicles.

References

[MPC] Model Predictive Control. https://github.com/WuStangDan/mpc-course-assignments.

[WIK] Model Predictive Control. https://en.wikipedia.org/wiki/Model_predictive_control.